DOCUMENT RESUME

ED 444 505 IR 020 161

AUTHOR Roessling, Guido; Freisleben, Bernd
TITLE Approaches for Generating Animations for Lectures.
PUB DATE 2000-00-00
NOTE 7p.; In: Society for Information Technology & Teacher Education International Conference: Proceedings of SITE 2000 (11th, San Diego, California, February 8-12, 2000). Volumes 1-3; see IR 020 112.
PUB TYPE Reports - Evaluative (142) -- Speeches/Meeting Papers (150)
EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Animation; *Audiovisual Aids; *Computer Software; Computer Uses in Education; *Material Development
IDENTIFIERS *Computer Animation

ABSTRACT

This paper provides a short review of the following basic approaches for generating animations so that teachers can determine the way best suited for them and be better prepared to select a tool addressing their needs: (1) using classical presentation tools such as PowerPoint; (2) visual editing using drag and drop or selection of options; (3) direct animation of source code; (4) using function calls implemented in a function library; and (5) animations generated by a scripting language or text commands. For each basic approach, several sample tools are cited. All selected tools are available for free use or download. (MES)

Approaches for Generating Animations for Lectures

Guido Rößling and Bernd Freisleben
Department of Electrical Engineering and Computer Science
University of Siegen, Germany
{roessling, freisleb}@informatik.uni-siegen.de

Abstract: Many software tools try to address the growing interest in supplementing course materials with animations. In this paper, we give a short review of the basic approaches for generating animations. Thus, teachers can determine the way best suited for them and are then better prepared to select a tool addressing their needs. For each basic approach, several example tools are cited. All selected tools are available for free use or download, so that interested teachers can test them.

Introduction

The number of teachers who want to integrate animations into their lectures to illustrate dynamic behavior has grown over the last few years. By now, a large selection of tools for generating or presenting animations are available. However, these tools differ in how animations are generated or edited, the offered animation features, portability and cost factors. Many people interested in using animations may find themselves confused by the profusion of different offers. This situation may lead to deciding on a tool not well suited to the intended animation content, resulting in increased time spent on animation generation and possibly frustration on the teacher's part.

In this paper, we try to help teachers wanting to use animations for supplementing their courses by outlining the general approaches of how animations can be generated. We give some examples of how generation efforts may typically look like, and contrast strengths and weaknesses of the approaches. For each general approach described, we also mention some of the tools supporting this approach.

Our interpretation of the very broad term animation is focused on instructional usage for computer science. Thus, typical animation areas contain the dynamic display of the effect commands or operations have on algorithms or data structures. We are not concerned with animations as seen in movies, like *"Antz"* or *"A Bug's Life"*. However, we include animations taken from other fields, such as physics, mathematics or economics.

The paper is organized as follows. In the next section, we mention some of the basic restrictions placed on animations tools. The following section discusses the presented basic approaches for generating animations and gives short examples for each approach. The last section concludes the paper and outlines areas for future research.

General Considerations and Restrictions

Apart from the generation process itself, some other issues should be taken into account before deciding on a tool. These issues are of a general nature and not directly linked to a given tool or generation approach. However, they can have a large impact on the usability of the tool and the generated animations, and should therefore be considered first.

The foremost issue from our point of view is platform independence. While the majority of students may own a PC and are most likely using Windows 95/98, this may not be the case for a significant portion of potentially interested clients. This is most especially true for universities, where Unix operating systems are very popular, as well as for the increasing number of Linux users. Furthermore, not everybody uses Intel-

compatible hardware – think, for example, of the *iMac* platform. Thus, using a proprietary tool based on Windows 95/98 may limit the number of potential clients for your software. Furthermore, the visibility and accessibility of the animations when propagated over the World-Wide Web may be reduced due to security concerns regarding ActiveX controls and JavaScript.

The same is true for formats that "only" require a plugin for Web browsers. Teachers should be aware that while the plugin download itself is usually free, plugins are not necessarily available for all platforms! Unix-based platforms and Linux are not supported by many tools. For maximum platform independence and the greatest possible audience, the chosen tool should provide a Java applet for displaying generated animations, as most modern Web browsers come with built-in Java support.

Furthermore, several of the commonly used tools are commercial software. Teachers should make sure that a player or viewer component is also included and may be freely passed along, so that students can study the animation at home whenever they want.

Basic Approaches for Animation Generation

The following basic approaches for animation generation are discussed in this paper:

- Using "classical" presentation tools such as PowerPoint®,
- Visual editing using drag and drop or selection of options,
- Direct animation of source code,
- Generating animations by function calls implemented in a function library (API),
- Animations generated by a scripting language or text commands.

Some of the tools available support more than a single way of generating animations, or use a special combination of the basic approaches. In the following, we discuss the individual approaches, mention some of the tools concerned and outline typical strengths and weaknesses.

Using "Classical" Presentation Tools

Some classical presentation tools like PowerPoint have lately been enhanced to support animation effects to a certain extent. Classically, many animation effects are concerned with modifying the appearance of new elements or the repositioning of visible components. The programs are not intended to be used for animating dynamic structures, like sorting algorithms or molecular movements. Thus, presentation tools are well suited for making presentations on slides more lively, but also restrict the teacher's creativity in generating animations.

The strengths of using presentation tools for animations include the comparative ease of using the presentation tools, as they typically offer visual editing and thus directly show the effects of a given action. Furthermore, they allow the integration of animations in lectures without having to switch media or presentation programs, as several courses are already presented in PowerPoint. However, the extent of possible animation effects supported by the tool limits their usage. Before spending a lot of time on trying to generate an animation, it is therefore reasonable to determine whether the effects offered are sufficient for the intended animation. If this is not the case, one can better spend the time otherwise wasted on trying out animation effects by using a tool better suited for the targeted animation.

Visual Editing Using Drag and Drop or Selection of Options

Probably the easiest way to generate animations is visual editing, where the teacher can directly see what the animation looks like at a given point in time. This approach is very similar to the way presentations are

generated in presentation tools such as PowerPoint. However, most animation tools offer special object types for certain contexts, such as arrays or list elements (Rößling & Freisleben 1999).

Object drawing is usually accomplished by one of the following approaches:

- dragging predefined objects on a drawing area, where they can be rotated or scaled,
- or by placement of individual points of a given object by a simple series of clicks.

Tools supporting this approach include *Macromedia Dreamweaver* (Macromedia 1999), *Flash* (Flash 1999) and *Animal* (Rößling & Freisleben 1999). The main advantage of this approach includes the direct visualization of effects already mentioned. Teachers can also learn much about animation generation by experimenting, as this can be accomplished by simply drawing objects to see whether they "fit" into the animation. Finally, only very modest computer experience and no programming experience is required for either animation generation or editing.

The main drawback of the approach is that, due to being forced to do everything manually, animation generation is very time-consuming. This becomes most obvious when more than a single animation for a given topic is wanted. Furthermore, there may be differences in what a given algorithm does and what is shown in the animation, as the animation depends on the author's understanding and interpretation of the algorithm, which may differ from the actual algorithm's behavior.

Direct Animation of Source Code

In some cases, teachers may have source code for the topic to be animated. In this case, an approach that can directly generate an animation from the source code would be highly helpful.

Tools such as *Jeliot* (Haajanen et al. 1999) and *AlgAE* (Zeil 1999) support direct source code animation. With these tools, the teacher is completely freed from having to define animation commands or doing extra work to generate the animation. The animation is also automatically kept consistent with the underlying source code. By being based directly on the source code, new animations can usually be generated by simply exchanging algorithm parameters and restarting the algorithm.

However, there are also some drawbacks. First of all, teachers must have access to source code for their problem in the programming language supported by the animation tool. The tools cited above support Java, while other tools others support Modula-2, Pascal, C or C++ code. If the teachers wish to use a programming language not supported by the chosen tool, the tool becomes unusable. Furthermore, some topics which may profit very much from animation can not easily be specified in source code. This may be due to the long running time of the problem, as for NP hard problems, or the lack of clarity and brevity of the code. In these cases, teachers need a way to directly jump to a given point in the animation, as well as a way to use pseudo code or a textual description replacing the actual source code. However, tools based on source code animation do not provide these features. Obviously, the teacher also needs to be a rather good programmer in the underlying language in order to generate animations. Thus, this approach is limited to rather special circumstances. Finally, the teacher often has no way to determine the way the code is presented, as the animation effects themselves are usually hard-coded into the underlying source code interpreter. Thus, teachers are prevented from choosing a different view of a given topic that they consider more interesting or beneficial.

Using Functions Calls Implemented in a Function Library

In order to alleviate the problems resulting from having a direct match between source code and animation, several tools support the generation of animations by function calls. These tools include *Animal* (Rößling & Freisleben 1999) and *JAL* (Silicon Graphics 1998). In this approach, animations are generated within a collection of functions provided in a function library for a specific output tool. The approaches can be

subdivided in two categories: tools which generate and directly display the animation, and tools that generate and store animation files that may be loaded and run at the teacher's discretion. JAL belongs to the first category, while Animal is part of the second category.

The functions are usually assembled in function libraries commonly called *APIs* (Application Programmers Interfaces). The API is responsible for providing functions that generate animations or animation files used by the underlying animation tool.

The main strength of this approach lies in the ease of generating animations "on the fly": once a given source code has been enriched with the corresponding API calls, it can be used to generate an arbitrary number of animations by simple substitution of the algorithm's parameters. The same is true for generated animations resulting from direct source code animation. However, API calls also allow teachers to define how the animation should look. Thus, the teacher need only decide *once* on a suitable way to present the animation of a given topic, and can then use this approach for generating animations of the topic whenever they are needed. Furthermore, the API calls can also be used for skipping "irrelevant" algorithm parts, allowing the teacher to focus on the main part of the presentation.

The main drawback of this approach lies in the need for teachers to be comfortable with programming. The API calls are usually embedded directly in the source code, or collected in a special animation generation class. This may prevent teachers with little or no programming knowledge in the underlying programming language from using the approach. Furthermore, the approach is only helpful if the teacher uses the same programming language for the course as used by the API, as calling functions in a library written in a different programming language remains difficult. Adapting the animation to other requirements or a slightly different display can become very time-consuming due to having to decide on how to change the API calls.

Animations Generated by a Scripting Language or Text Commands

Animation tools working with scripting languages try to combine the flexibility of API calls with a modest amount of programming knowledge. The scripting language may be a proper but "easy" language, or it may simply be restricted to specific text commands without programming structures such as conditional statements or loops. Tools supporting animation generation by a scripting language or text commands include *JAWAA* (Rodger 1997), *JSamba* (Stasko 1998) and *Animal* (Rößling & Freisleben 1999). Usually, the commands can be generated and edited with any text editor and contain a single command per line. Commands are usually defined in an "intuitive" way, making the animation definition (almost) self-explanatory. Thus, most teachers will find it very easy to read animation files defined by this approach even without having programming knowledge.

The main strength of the approach lies in the fact that the process of generating the animation definition can also be automated. Doing so yields results similar to the effect of using API calls, but without being forced to use a certain programming language. Furthermore, due to the normally good readability of the animation code lines, many teachers may find this approach easier to use than API calls.

However, while each single line of the generated animation may be easily readable, it is often rather difficult to get a feeling for what the animation might look like. While experimenting with the code lines is easy, generalizing the desired effects and putting them into algorithm's source code can be difficult. This is necessary, however, for allowing the generation of animations "on the fly" by adapting algorithm parameters.

Conclusions

In this paper, we have outlined several possible general approaches for generating animations. Apart from the basic considerations such as platform independence and free availability, the approaches mainly differ in three aspects: (I) the way animations are generated, (II) the extent to which the actual display can be

adapted to the teacher's wishes, and (III) how easily multiple animations of the same topic can be generated.

Due to limitations inherent in the target use of *presentation programs*, we can not recommend them for arbitrary animations. *Visual editing* offers a very easy way of generating animations and highly flexible display depending on the underlying tool's abilities, but does not allow for easy adaptation of a given animation to modified input values. *Direct animation of source code* offers the easiest animation generation, as this is done by the animation system. Generating multiple animations of the same topic is also done without effort. However, the teacher is usually not able to adapt the default display to his or her preferences. Furthermore, the programming language to be used for all examples is determined by the animation tool, and "irrelevant" passages of the code cannot easily be skipped. *Function calls* in a special animation generation function library, on the other hand, allow teachers to adapt the display and easily generate multiple animation, but also puts demands on their programming ability as well as the underlying programming language. *Scripting languages or text commands* offer a middle way, combining easy generation and high. However, trying to automate the generation can become difficult, especially for teachers with little programming knowledge.

In general, we want to stress that tailoring the animations to the specific needs is usually a very time-consuming process. Thus, it is of great interest to be able to automate parts of the process, so that the same steps need only be performed once for a given topic, and can then easily be reused with changed parameter values. Most approaches offer some way to automate this process; however, this is not true for the most "intuitive" approaches taken by *presentation tools* and *visual editing*.

In the end, it is up to the teacher to decide which type of tool is best suited for the current task, possibly changing the tool used from one week to the next. All major approaches have their specific strengths and weaknesses, so that teachers should study the effects of the tool's approach before deciding on a given tool. This may well result in the teacher using a different tool each week, though this has severe drawbacks in the form of the preparation time for getting familiar with the tool. Therefore, teachers should select tools supporting more than a single technique of generating or editing animations.

The animation tool *Animal* (Rößling & Freisleben, 1999) developed by us supports animation generation by *visual editing, API calls* and a *scripting language*. It can thus be used by teachers of any background and easily adapted to the current requirements by simply switching the way animations are generated. The basic graphic elements point, polyline, arc, text and list element are supported and can easily be adapted to more specific structures such as circle segments. All objects can be rotated, moved, shown/hidden or change color using a duration and/or offset defined by the teacher. For presentations, snapshots of the current animation state can be generated by the animation player and are stored in a variety of popular graphics formats such as JPEG or PNG. *Animal* is implemented completely in Java and freely available for all platforms supporting Java at the URL http://www.informatik.uni-siegen.de/~inf/Software/Animal.

References

Flash (1999). *Home Page of Macromedia's Flash Tool*: http://www.macromedia.com/software/flash/index.html

Haajanen, J., Pesonius, M., Sutinen, E., Tarhio, J., Teräsvirta, T., Vanninen, P. (1997). Animation of User Algorithms on the Web. *IEEE Symposium on Visual Languages, 1997*, IEEE. 360-367.

Macromedia (1999). *Home page of Macromedia's Dreamweaver Tool*: http://www.macromedia.com/software/dreamweaver/index.html

Rodger, S. (1997). *JAWAA and Other Resources*. Available on the WWW: http://www.cs.duke.edu/~rodger/tools/tools.html

Rößling, G., & Freisleben, B. (1999). *Animal – A New Interactive Modeler for Animations in Lectures.* ACM SIGCSE 2000 Proceedings (to appear).

Silicon Graphics Inc. (1999). *JAL Algorithm Animation.* Available on the WWW at URL http://reality.sgi.com/austern/java/demo/demo.html

Stasko, J. (1998). *Samba Algorithm Animation System.* Available on the WWW: http://www.cc.gatech.edu/gva/softviz/algoanim/samba.html

Zeil, S. J. (1999). *AlgAE: Algorithm Animation Engine v2.0.* Available on the WWW: http://www.cs.odu.edu/~zeil/algae.html

U.S. Department of Education
Office of Educational Research and Improvement (OERI)
National Library of Education (NLE)
Educational Resources Information Center (ERIC)

NOTICE

REPRODUCTION BASIS

☒ This document is covered by a signed "Reproduction Release (Blanket) form (on file within the ERIC system), encompassing all or classes of documents from its source organization and, therefore, does not require a "Specific Document" Release form.

☐ This document is Federally-funded, or carries its own permission to reproduce, or is otherwise in the public domain and, therefore, may be reproduced by ERIC without a signed Reproduction Release form (either "Specific Document" or "Blanket").